

nmat: un sistema client server interattivo per operazioni matriciali

Susanna Pelagatti

Progetto di recupero del corso di LPS 2006/07

Indice

1	Introduzione	
1.1	Materiale in linea	
1.2	Struttura del progetto e tempi di consegna	
1.3	Valutazione del progetto	
2	Il progetto: nmat	
2.1	Specifiche del server	
2.2	Specifiche del client	
2.3	Gestione dei dati dei diversi utenti .	
2.4	Comunicazioni da nmatc a nmats . .	
2.5	Comunicazioni da nmats a nmatc . .	
2.6	Elaborazione del server e del client .	
2.7	Richiesta di autorizzazione	
2.8	Comandi di nmatc	
2.8.1	Creazione matrice (=)	
2.8.2	Stampa matrice (print)	
2.8.3	Matrici correnti (list)	
2.8.4	Salva su file (save)	
2.8.5	Carica da file (load)	
2.8.6	Fine sessione (exit)	
2.8.7	Aiuto in linea (help)	
2.8.8	Trasposta (TRASP)	
2.8.9	Operazioni matriciali (+ - *)	
3	Test automatici	
4	Codice e documentazione	
4.1	Vincoli sul codice	
4.2	Formato del codice	
4.3	Relazione	

1 Introduzione

1 Il progetto consiste nello sviluppo di un sistema client-server che realizza operazioni su matrici e dei relativi programmi di test.

1.1 Materiale in linea

Il programma dell'ultimo corso di LPS può essere reperito su Web alla pagina

<http://www.di.unipi.it/~susanna/LPS/>

si chiede esplicitamente di studiare la parte di programmazione di sistema (File, Pipe, Processi e Segnali) sul testo di LCS 06/07 e di fare riferimento ai lucidi di LCS 06/07 relativi a questi argomenti reperibili su

<http://www.cli.di.unipi.it/doku/doku.php>

Inoltre altre informazioni interessanti sono reperibili alla pagina degli avvisi ed alle FAQ del medesimo sito.

Eventuali chiarimenti possono essere richiesti consultando i docenti di LCS durante l'orario di ricevimento, le ore in laboratorio e/o per posta elettronica.

1.2 Struttura del progetto e tempi di consegna

6 Il progetto deve essere sviluppato da un singolo studente individualmente e può essere consegnato entro il 1 Febbraio 2008.

6 La consegna del progetto avviene *esclusivamente* per posta elettronica, attraverso il target **consegna** del **Makefile** contenuto nel kit di sviluppo del progetto.

Questo forza la seguente struttura per il progetto: un'unica directory `nmat/` che contiene (almeno) i seguenti file:

- un file `README` che include una spiegazione dettagliata del contenuto dei vari file e le indicazioni per l'utente (compilazione, installazione, esecuzione, bug noti etc.)
- almeno due file `.c`:
 - `nmats.c` che contiene il main del server ed eventuali funzioni ausiliarie
 - `nmatc.c` che contiene il main del client ed eventuali funzioni ausiliarie
- eventuali altri file `.c` e `.h` relativi alle librerie sviluppate e ai loro header.
- un file `Makefile` che contenga almeno i seguenti target: `all` che genera gli eseguibili per il server ed il client; e `clean` che elimina i file oggetto (`.o`)
- la relazione sul lavoro svolto in formato PDF.
- un file `gruppo.txt` che specifica numero di matricola, cognome, nome ed indirizzo di mail dell'autore, secondo il formato (notare la virgola come separatore)

Zini, Gino, 123456, zini@madoc.it

Tutti i file del progetto si devono trovare nella directory `nmat/` e non sono ammesse sottodirectory.

*I progetti che non rispettano il formato o non consegnati con il target **consegna** non verranno accettati.*

Si noti inoltre che gli studenti che intendono discutere il progetto entro Luglio 2007 devono consegnarlo entro e non oltre il 30/06/07, gli studenti che intendono discutere il progetto entro Settembre 2007, devono consegnarlo entro e non oltre il 10/09/07, mentre la data ultima di consegna per gli altri è il 01/02/08.

1.3 Valutazione del progetto

Al progetto viene assegnato un punteggio da 0 a 30 in base ai seguenti criteri:

- motivazioni, originalità ed economicità delle scelte progettuali

- strutturazione del codice (suddivisione in moduli, uso di makefile e librerie etc.)
- efficienza e robustezza del software
- aderenza alle specifiche
- qualità del codice C e dei commenti
- chiarezza ed adeguatezza della relazione

La discussione del progetto tenderà a stabilire se lo studente è realmente l'autore e verterà su tutto il programma del corso. Il voto dell'orale (ancora da 0 a 30) fa media con la valutazione del progetto per delineare il voto finale.

Casi particolari Agli studenti iscritti ai vecchi ordinamenti (nei quali il voto di LPS contribuiva al voto di SO) sarà assegnato un voto in trentesimi che verrà combinato con il voto del corso di SO corrispondente secondo modalità da richiedere ai due docenti coinvolti.

2 Il progetto: nmat

Lo scopo del progetto è lo sviluppo di due programmi C, un server `nmats` e un client `nmatc`, per il calcolo di operazioni su matrici e dei relativi programmi di test (sez. 3).

Il server implementa un insieme di operazioni su matrici (somma, trasposta etc.) assieme ad alcune funzioni di utilità come il salvataggio su file delle matrici, la visualizzazione di una matrice, ecc.). Il client implementa l'interfaccia fra l'utente ed il programma come un *interprete di comandi*: una volta attivato interagisce con l'utente tramite una linea di comando digitata da tastiera.

L'interazione avviene nel seguente modo. Il server viene attivato e si mette in ascolto di richieste di connessione da parte di processi client su una pipe con nome fissato. Quando una richiesta di connessione viene ricevuta, il server crea un processo figlio che interagirà con il client che ha richiesto la connessione (fino alla terminazione del client), ed invia un messaggio di avvenuta connessione al client.

Una volta ricevuto il messaggio di avvenuta connessione, il client si mette in attesa di comandi da parte dell'utente e traduce tali comandi in opportune richieste al server figlio, e visualizza le risposte ottenute.

Server e client sono residenti sulla stessa macchina e quindi possono interagire correttamente utilizzando named pipe.

Si assume inoltre che in ogni momento l'utente possa lavorare su un *insieme illimitato* di matrici. Ogni matrice è composta da un numero illimitato di righe e colonne e può contenere elementi di tipo **double**. Inoltre ogni matrice è univocamente identificata da una stringa di lunghezza compresa fra 1 e 8 caratteri alfanumerici. Il primo carattere deve essere una lettera.

2.1 Specifiche del server

Il server **nmats** viene attivato digitando

```
bash:~$ nmats userlist
```

dove **userlist** è un file di testo con i nomi degli utenti che possono richiedere connessioni con il server, uno per linea. Ad esempio:

```
bash:~$ more userlist
pippo
pluto
paperone
minnie
bash:~$ more userlist
```

Una volta attivato, **nmats** lavora in background finché non viene terminato dall'esterno con un segnale SIGTERM.

Alla ricezione del segnale, il server termina gentilmente inviando un messaggio di terminazione a tutti gli utenti connessi. Inoltre, prima di terminare, ripulisce la directory corrente rimuovendo le pipe utilizzate per la comunicazioni e le informazioni temporanee relative agli utenti connessi.

2.2 Specifiche del client

Il client viene attivato da shell digitando

```
bash:~$ nmatc user
```

All'avvio, **nmatc** chiede al server **nmats** l'autorizzazione per la connessione. L'autorizzazione viene concessa solo se **user** è uno dei nomi contenuti nel file **userlist** e se non esistono altre connessioni attive per lo stesso utente. Se l'utente non viene autorizzato, l'esecuzione di **nmatc** termina visualizzando un messaggio di errore sullo standard error.

Se viene autorizzato, **nmatc** visualizza un opportuno prompt

```
bash:~$ nmatc utente
Welcome in nmatc!
-?
```

ed entra in un ambiente interattivo che permette l'esecuzione di un insieme di comandi. Nella Sezione 2.8 per ogni comando accettato, è specificata la sintassi, il significato, e alcune delle condizioni in cui è richiesto di segnalare un errore. Ulteriori condizioni di errore devono essere fissate dallo studente durante la stesura del progetto e specificate nella documentazione allegata.

Il client termina gentilmente alla ricezione di un SIGINT. In questo caso rimuove la pipe utilizzata per colloquiare con il server e stampa un breve messaggio su stdout

```
nmatc terminating ....
bash:~$
```

2.3 Gestione dei dati dei diversi utenti

Il programma server **nmats** appena attivato crea una pipe con nome **clientserver** nella directory corrente. Inoltre, sempre nella directory corrente, viene creata la directory

```
nmat_data/
```

se non è già presente. In **nmat_data**, viene creata una directory per ogni utente che compare nel file **userlist**, se non è già presente, ad esempio

```
bash:~$ ls nmat_data
pippo/
pluto/
paperone/
minnie/
```

Durante l'esecuzione, in ogni sottodirectory vengono tenuti i file temporanei prodotti dalle operazioni richieste dall'utente corrispondente.

2.4 Comunicazioni da nmatc a nmats

Ogni volta che un utente attiva un nuovo client, questi invia una richiesta di connessione sulla pipe **./clientserver** (nb: assicurarsi che alla

terminazione di tutti i client il server rimanga attivo!).

Le richieste di connessione a **nmats** hanno tutte il seguente formato:

PID CONNECT user

dove **user** è il nome dell'utente che richiede la connessione e **PID** è il PID del processo client che ha inviato la richiesta. Se la connessione è accettata il server crea nella directory corrente una pipe di nome **./clientserverPID** (dove **PID** sta per il valore del PID appena ricevuto). Tale pipe servirà per tutte le comunicazioni successive del client con il server. Tale pipe viene eliminato quando il client si disconnette o comunque alla terminazione del server.

Quando l'utente richiede una nuova operazione al proprio client **nmatic**, questi la trasforma in una richiesta al processo **nmats**. Le richieste vengono inviate su **./clientserverPID** ed hanno il formato:

CODICEOP ARGOMENTI

dove **CODICEOP** è una sequenza di caratteri alfanumerici che identifica l'operazione da eseguire e **ARGOMENTI** sono gli argomenti necessari a portare a termine la richiesta.

I codici ed il formato delle varie operazioni devono essere fissati in fase di stesura del progetto in modo da avere una efficace decodifica ed adeguatamente documentati nella relazione.

2.5 Comunicazioni da nmats a nmatic

Quando arriva una richiesta da **nmatic**, **nmats** la elabora e poi invia una risposta sul pipe

./serverclientPID

dove **PID** è il process identifier del client in questione. Questo pipe viene creato dal client al suo avvio o comunque prima di inviare la richiesta al server. Al momento della terminazione, il client si preoccupa anche di cancellare la propria pipe **./serverclientPID**.

Il formato della risposta dipende dalla richiesta effettuata e viene lasciato come scelta di progetto. Anche in questo caso è richiesta una chiara documentazione dei formati scelti nella relazione finale.

2.6 Elaborazione del server e del client

Il client trasforma le richieste dell'utente in messaggi per il server. Il server alla ricezione di una richiesta effettua l'elaborazione e risponde al client. Il client visualizza infine le risposte all'utente.

Le richieste che possono essere inviate dal client sono essenzialmente di due tipi:

- richiesta di autorizzazione ad iniziare le operazioni : inviata quando **nmatic** viene attivato
- richieste di operazioni su matrici

descriveremo le varie tipologie nelle seguenti sottosezioni.

2.7 Richiesta di autorizzazione

All'avvio, **nmatic** genera una richiesta sul pipe **./clientserver** di formato

PID CONNECT user

come descritto precedentemente.

Alla ricezione di una richiesta di **CONNECT**, il server verifica che il nome simbolico dell'utente corrispondente a **user** compaia nel file di controllo degli accessi.

Se l'utente che ha attivato il client ha il diritto di farlo e non è già connesso, il server attiva un figlio che si occupa di interagire con il client identificato da **PID** da questo punto fino alla terminazione del client stesso. L'avvenuta connessione è segnalata al client da un messaggio sul pipe **./serverclientPID**. Altrimenti, se l'utente non ha diritto di connettersi, il server invia un messaggio di rifiuto sullo stesso pipe (**./serverclientPID**).

2.8 Comandi di nmatic

L'interprete di **nmatic** accetta dall'utente l'insieme di comandi descritti nel seguito.

2.8.1 Creazione matrice (=)

Comando =:

nomemat = val11 val12 ... val1n; val21 val22 ... val2n; valm1 valm2 ... valmn;

Descrizione:

Permette di aggiungere all'insieme di matrici la matrice *nomemat* oppure, quando già presente, di modificarne il contenuto. Ogni elemento della matrice è separato dal successivo da uno (o più) spazi; ogni riga è separata dalla successiva da un ';'. Le righe devono avere un numero eguale di elementi. L'assenza di elementi a destra del segno di uguale corrisponde alla *cancellazione* della matrice dall'insieme.

Errori:

Devono essere segnalati errori nel caso di disuniformità delle righe e in tutti gli altri casi di inammissibilità della matrice fornita. Inoltre deve essere segnalato un errore nel caso in cui non sia possibile aggiungere la matrice all'insieme delle matrici già esistenti.

Esempio:

```
-? Tmp = 1 1.2 1.3 2; 1 12 15 1.5; 1
1.3 2 130;
-? Prodotto =
crea una matrice Tmp con tre righe e 4 colonne
e cancella la matrice Prodotto liberando la
memoria allocata. Non è possibile modifica-
re le dimensioni di una matrice attraverso
l'assegnamento se la matrice non e' stata
presedentemente cancellata. Nel nostro caso:
-? Tmp = 0;
Error: unable to change matrix size
-? Prodotto = 0; 1.2;
dove l'assegnamento a Prodotto ha avuto
successo.
```

2.8.2 Stampa matrice (print)**Comando print:**

```
print nomemat
```

Descrizione:

Visualizza il contenuto della matrice *nomemat* (per righe separate da una andata a capo).

Esempio:

```
-? print Tmp
1 1.2 1.3 2
1 12 15 1.5
1 1.3 2 130
```

2.8.3 Matrici correnti (list)**Comando display:**

```
display
```

Descrizione:

Visualizza l'elenco dei nomi delle matrici presenti nell'insieme e la loro dimensione (Righe; Colonne), secondo il formato specificato nell'esempio:

Esempio:

```
-? display
Tmp(3;4)
Prodotto(1;2)
```

2.8.4 Salva su file (save)**Comando save:**

```
save nomefile
```

Descrizione:

Salva l'insieme delle matrici (nome, dimensioni, valori) sul file *nomefile*. Durante la fase di salvataggio è necessario verificare l'esistenza di *nomefile* e, se il file esiste, chiedere conferma per la riscrittura. Nella documentazione, si descriva il formato del file adottato, motivandone le scelte.

Esempio:

```
-? save miofile
miofile saved
```

2.8.5 Carica da file (load)**Comando load:**

```
load nomefile
```

Descrizione:

Carica un insieme di matrici (nome, dimensioni, dati) dal file *nomefile* verificando se sono già presenti in memoria di lavoro. Nel caso in cui una matrice del file sia già presente in memoria, chiede all'utente conferma per la riscrittura.

Errori:

Deve essere segnalato un errore nel caso in cui non sia possibile aggiungere una matrice all'insieme delle matrici.

Esempio:

```
-? load miofile
miofile loaded
```

2.8.6 Fine sessione (exit)**Comando exit:**

```
exit
```

Descrizione:

Provoca la terminazione del client e la rimozione del pipe di risposta `./serverclientPID`. Anche il figlio del server che sta interagendo con `nmactc` deve terminare correttamente. Se esistono matrici non precedentemente salvate, si deve chiedere conferma all'utente.

Esempio:

```
-? exit
```

2.8.7 Aiuto in linea (help)**Comando:**

```
help
```

Descrizione:

Visualizza l'elenco dei comandi disponibili in `nmactc` e la loro sintassi, descrivendo brevemente il loro significato.

2.8.8 Trasposta (TRASP)**Comando TRASP:**

```
nomemat1 = TRASP nomemat2
```

Descrizione:

Esegue la trasposta della matrice *nomemat2*. La matrice risultato *nomemat1* viene aggiunta all'insieme delle matrici oppure, se già presente, vengono modificati i valori dei suoi elementi.

Errori:

Deve essere segnalato un errore nel caso in cui non sia possibile aggiungere la matrice all'insieme delle matrici.

Esempio:

```
-? TraspTmp = TRASP Tmp
assegna la trasposta di Tmp a TraspTmp.
```

2.8.9 Operazioni matriciali (+ - *)**Comandi + - *:**

```
nomemat = nomemat1 op nomemat2
dove op può essere solo +, - oppure *.
```

Descrizione:

Esegue l'operazione indicata sulle matrici operandi *nomemat1* e *nomemat2*. La matrice risultato *nomemat* viene aggiunta all'insieme delle matrici oppure, se già presente, vengono modificati i valori dei suoi elementi.

Errori:

Devono essere segnalati errori nel caso di incompatibilità dell'operazione a causa dell'incompatibilità degli operandi. Inoltre deve essere segnalato un errore nel caso in cui non sia possibile aggiungere la matrice all'insieme delle matrici.

Esempio:

```
-? Prodotto = Tmp * TraspTmp
-? Somma = matrice1 + matrice2
-? Diff = matrice1 - matrice2
-? matrice1 = matrice1 - matrice2
```

3 Test automatici

Lo studente deve sviluppare uno o più programmi di test che verifichino automaticamente il funzionamento del server e del client.

L'invocazione di tutti i test deve avvenire automaticamente attraverso il target `test` del `makefile`.

Il funzionamento dei programmi di test, il loro utilizzo e la copertura del codice ottenuta¹ Devono essere specificati nella relazione finale.

4 Codice e documentazione

In questa sezione, vengono riportati alcuni requisiti del codice sviluppato e della relativa documentazione.

4.1 Vincoli sul codice

Makefile e codice devono compilare ed eseguire **CORRETTAMENTE** su (un sottinsieme non vuoto

¹Usare `gcov` o `lcov` come illustrato in laboratorio.

del) le macchine del CLI. Il README deve specificare su quali macchine è possibile far girare correttamente il codice. Inoltre, se si usano software e librerie non presenti al CLI: (1) devono essere presenti nel tar TUTTI i file necessari per l'installazione in locale del/i tool e (2) devono essere presenti nel makefile degli opportuni target per effettuare automaticamente l'installazione in locale. Se questa condizione non è verificata il progetto non viene accettato per la correzione.

Inoltre la stesura del codice deve osservare i seguenti vincoli:

- la compilazione del codice deve avvenire definendo delle regole appropriate nella parte iniziale del makefile contenuto nel kit;
- il codice deve compilare senza errori o warning utilizzando le opzioni `-Wall -pedantic`
- NON devono essere utilizzate funzioni per la manipolazione delle stringhe che *non* limitano il numero di caratteri scritti/manipolati, ad esempio la `strcpy()` deve essere evitata a favore della `strncpy()` in cui è possibile fissare il massimo numero di caratteri copiati (per le motivazioni consultare il man in linea)
- NON devono essere utilizzate funzioni di temporizzazioni quali le `sleep()` o le `alarm()` per risolvere problemi di race condition o deadlock fra i processi. Le soluzioni implementate devono necessariamente funzionare qualsiasi sia lo scheduling dei processi coinvolti
- DEVONO essere usati dei nomi significativi per le costanti nel codice (con opportune `#define` o `enum`)

4.2 Formato del codice

Il codice sorgente deve adottare una convenzione di indentazione e commenti chiara e coerente. In particolare deve contenere

- una intestazione per ogni file che contiene: il nome ed il cognome dell'autore, la matricola, il nome del programma; dichiarazione che il programma è, in ogni sua parte, opera originale dell'autore; firma dell'autore.

- un commento all'inizio di ogni funzione che specifichi l'uso della funzione (in modo sintetico), l'algoritmo utilizzato (se significativo), il significato delle variabili passate come parametri, eventuali variabili globali utilizzate, effetti collaterali sui parametri passati per puntatore etc.
- un breve commento che spieghi il significato delle strutture dati e delle variabili globali (se esistono);
- un breve commento per i punti critici o che potrebbero risultare poco chiari alla lettura
- un breve commento all'atto della dichiarazione delle variabili locali, che spieghi l'uso che si intende farne

4.3 Relazione

La documentazione del progetto consiste nei commenti al codice e in una breve relazione (massimo 10 pagine) il cui scopo è quello di descrivere la struttura complessiva del lavoro svolto. La relazione deve rendere comprensibile il lavoro svolto ad un estraneo, senza bisogno di leggere il codice se non per chiarire dettagli implementativi. In pratica la relazione deve contenere:

- le principali scelte di progetto (strutture dati principali, algoritmi fondamentali e loro motivazioni)
- la strutturazione del codice (logica della divisione su più file, librerie etc.)
- la struttura del server e del client
- la struttura dei programmi di test
- l'interazione fra client e server ed i relativi protocolli
- le difficoltà incontrate e le soluzioni adottate
- quanto altro si ritiene essenziale alla comprensione del lavoro svolto
- istruzioni per l'utente su come compilare/eseguire ed utilizzare il codice (in quale directory deve essere attivato, quali sono le assunzioni fatte etc) (da riportare anche nel README)

La relazione deve essere in formato .pdf.